Программирование

Python

Структуры данных. Классы

Кафедра ИВТ и ПМ

2018

План

Прошлые темы

Пользовательские структуры данных

- Концепция записи
- Запись как кортеж
- Классы

Outline

Прошлые темы

- Что такое тип?
- Какие простые типы есть в Python?
- Какие составные типы есть в Python?

Outline

Логически связанные данные

Иногда требуется работать одновременно с несколькими отдельными но логически связанными переменами.

Например координаты точки на плоскости представляются двумя переменными.

Это вызывает большие неудобства если таких логически связаных переменных становится много. Например описания погоды (температура, облачность, осадки) в определённом городе, в определённый день.

Неудобства проявляются особенно сильно если нужно создавать ещё одну группу таких переменных для хранения данных, например для другого города.

Outline

Запись

Логически такие наборы данных можно описывать в виде набора полей.

Предположим требуется хранить данные о погодных условиях (температура, облачность, осадки) в определённом городе, в полдень на определённую дату.

```
Данные о погоде:
    Город
    ДеньГода
    Температура
    Облачность
    Осадки
```

Такое представление логически сгруппированных данных будем называть *записью*.

Концепция записей

ВЫПИСЬ ИЗЪ МЕТРИЧЕСКОЙ КНИГИ, ЧАСТЬ ПЕРВАЯ, О РОДИВШИХСЯ, ЗА ГОДЪ.

Счетъ родившихся: Мужеска пола	Женска пола	Мѣсяцъ и день: рожд.	креще.	Имена родившихся.	Званіе, имя, отчество и фамилія родителей, и какого вѣроисповѣданія.	Званіе, имя, отчество и фамилія воспріемниковъ.	Кто совершалъ таинство крещенія.	Рукоприкладство свидетелей записи по желанію.
—	41.	Октября 25	31	Марія	Села Ново-Георгіевки приходской Священникъ Николай Тихоновъ Синайскій и законная его жена Елизавета Михайлова, оба православные.	Села Михайловскаго Священникъ Павелъ Александровъ Ильинскій и села Ново-Николаевскаго Священническая жена Екатерина Стефанова Николаевская.	Священникъ села Михайловскаго Павелъ Ильинскій съ діакономъ Василіемъ Муратовымъ	

1917 г. 25 ...

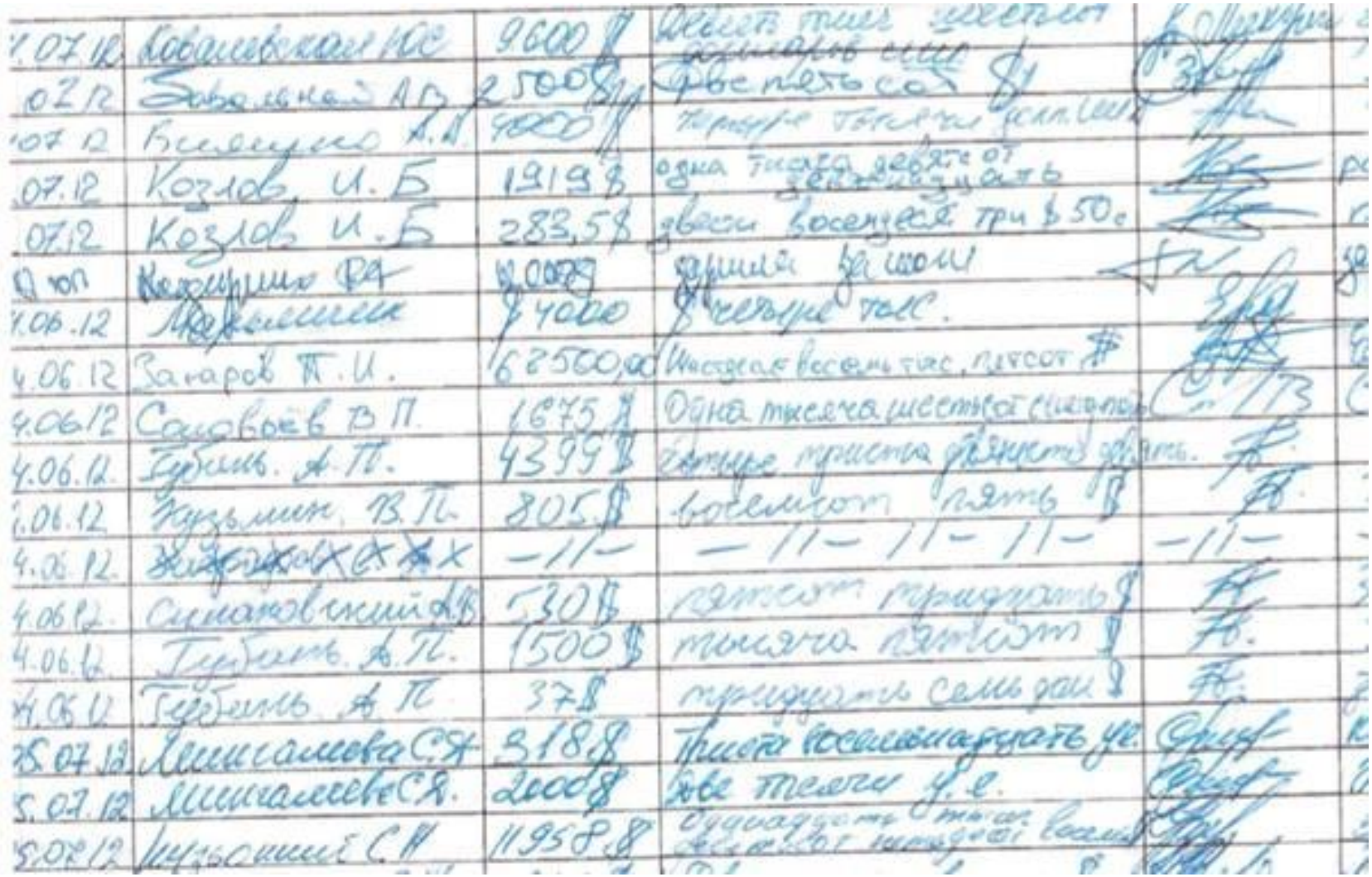
1917 г. Іюля 25 дня № 179

Сія метрическая выпись во всемъ согласна съ копіею метрической книги за тысяча восемьсотъ девяносто четвертый годъ, ч. 1я, хранящейся при храмѣ села Ново-Георгіевки, что удостовѣряемъ подписомъ и печатью.

Священникъ Николай Нечаевъ

Псаломщикъ ...

Концепция записей



Запись

Запись (record) — тип данных, набор значений различных типов.

Запись состоит из **полей**. Поле как было отмечено может быть представлено отдельным типом. В поле в том числе может быть другой записью.

Например поле *Дата* представляет собой отдельную запись состоящую из трёх полей.

```
Данные о погоде:
    Город
    Дата
        год
        месяц
        день
    Температура
    Облачность
    Осадки
```

Запись

В языках программирования записи могут быть представлены разными способами.

В Pyhton запись можно представить с помощью одного из составных типов данных: *кортежа, списка или словаря*.

Другой способ представления записи - тип данных **класс (class)**.

Хотя понятия класса включает в себя ещё и методы (операции производимые с данными), его можно использовать только для хранения данных.

Outline

Записи с помощью кортежей

Представим дату (год, месяц, день) как кортеж.

```
date1 = (2018, 03, 14)

#Теперь можно обращаться к отдельным элементам кортежа
date1[0]  # 2018
date1[2]  # 14

# Создавать списки из кортежей
dates = []
for i in range(10):
    d = randint(2000, 2018), \
        randint(1, 12),  \
        randint(1, 28)
    dates += [ d ]
```

Outline

Классы и объекты

Класс — составной **тип данных**, который может быть описан программистом.

Объект - экземпляр класса; *переменная* типа класс.

Классы и объекты

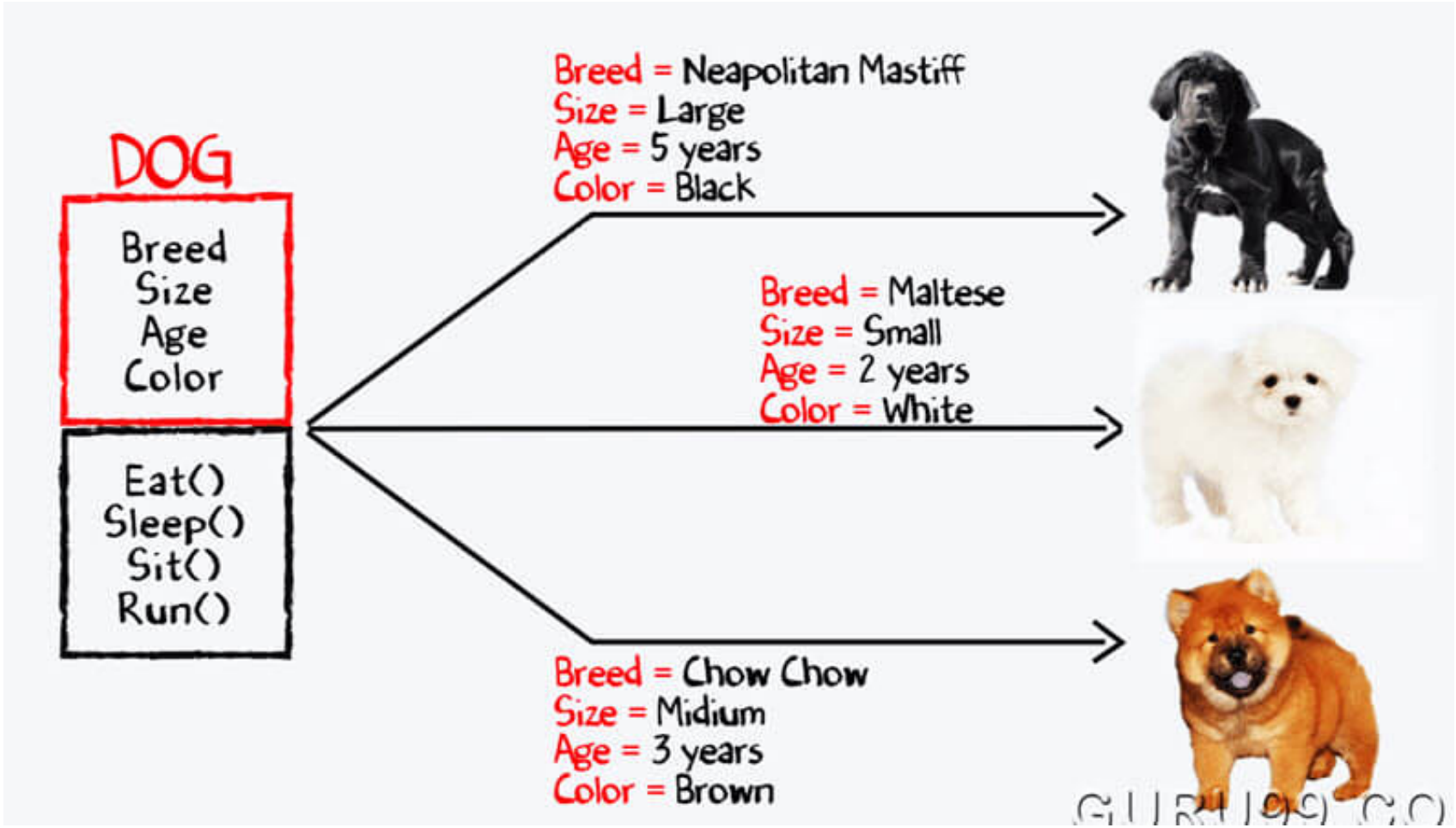
Классы могут включать в себя переменные других типов - **поля** (свойства).

Кроме того, класс может включать в себя функции - **методы**.

Класс как тип данных представляет собой только набор полей которым не заданы конкретные значений и набор методов.

Во время создания объекта (объявления переменной типа класс) этим полям задаются конкретные значения.

Классы и объекты



Класс как контейнер данных

Представим запись *Дата* в виде класса (создадим новый тип данных).

```
class Date:
    day = 1
    month = 1
    year = 1
```

В Python нельзя объявить поле, только определить: привести идентификатор и значение.

Когда будет создана переменная типа *Date* её поля уже будет содержать указанные при определение класса значения.

Если нужно отличать переменную с заполненными значениями полей, от аналогичной незаполненными то в качестве начальных значений используют None.

Класс как контейнер данных

Создание переменной описанного типа Date.

Переменная типа класс называется **объектом** или **экземпляром класса**.

```
my_birthday = Date()
othder_date = Date()
d1 = Date()


type( d1 )  # __main__.Date
```

Каждый описанный класс представляет собой отдельный тип данных.

С технической точки зрения имя класса в python включает ещё и имя пространства имён. В приведённом примере пространство имён называется __main__ [1]

[1] Имена переменных начинающиеся и заканчивающиеся символа подчёркивания играют роль служебных ("для внутреннего пользования"), и непосредственное использование таких имён не рекомендуется

Класс как контейнер данных

Чтобы получить доступ к полям используется **селектор** - оператор "."(точка).

```
my_birthday = Date()

my_birthday.day = 31
my_birthday.month = 1
my_birthday.year = 1956
```

Класс как контейнер данных

В Python любой тип являются классом, в том числе простые типы.

Однако часто с точки зрения программиста работа с переменными встроенных типов (int, float, tuple и т.д.) выглядит точно так же как и с обычными переменными (в других языках).

Класс как контейнер данных

```
my_birthday = Date()

my_birthday.day = 31
my_birthday.month = 1
my_birthday.year = 1956

# отдельные экземпляры класса независимы
d = Date()
d.year = 1984

print( my_birthday.year )  # 1956
print( d.year )  # 1984
```

Таким образом для логически связанных наборов данных следует создавать либо классы либо организовывать их с помощью встроенных типов: списков, кортежей, словарей.

Такой подход позволяет логически организовать данные, уменьшить количество отдельных переменных.

Точка на плоскости:

```
# точка 1
x1, y1 = 0,0

# точка 2
x2, y2 = 0,0

# 2 точки - 4 переменных
```

```
class Point2D:
    x = 0
    y = 0

# точка 1
p1 = Point2D()

# точка 2
p1 = Point3D()

# 2 точки - 2 переменные
```

Конструктор

Если используется кортеж вместо класса, то возможно задать значения элементов в одну строчку:

```
x,y = 7.2, -0.5
```

Однако если использовать этот способ для класса Point2D то будет создан кортеж

```
p = Point3D()
p = 7.2, -0.5

type( p )  # -> tuple
```

Конструктор

Чтобы решить эту проблему в класс, помимо *полей* нужно добавить *методы*, т.е. функции класса.

Метод который используется для создания объекта называется **конструктором**.
В Python такой метод должен всегда называться __init__

```
class Point2D:
    x = 0
    y = 0

    def __init__(self, x,y):
        self.x = x
        self.y = y

# теперь можно создать объект так
p = Point2D( 7.2, -0.5 )
```

Конструктор

- Конструктор является специальным методом и вызывается не так как остальные.
- Конструктор вызывается во время создания объекта (переменной)
- Конструктор инициализирует объект - задаёт начальные значения полей.
- Конструктор без параметров задаёт полям те значения, которые были приведены при их описании.

Конструктор

- Конструктор при описании внутри класса должен быть назван __init__, однако при его вызове должно быть указано иям класса:

```
def Point2D():
    x,y, = 0, 0
    def __init__(self, x,y):
        self.x = x
        self.y = y

p1 = Point2D(10, 2)
```

- Конструктор вызывается во время создания объекта

```
p1 = Point2D(10, 2)
```

Конструктор

```
class Point2D:
    x = 0
    y = 0

    def __init__(self, x,y,z):
        self.x = x
        self.y = y
```

Поля x и y класса Point2D не входят в область видимости метода __init__, однако им должны задаватся значения.

В Python к полям класса изнутри методов класса можно обращаться через переменную self.

Эта переменная представляет *сам объект*, описывается как первый параметр каждого метода, но при вызове метода передаётся неявно.

Другие методы

```
class Point2D:
    x = 0
    y = 0

    def r(self):
    """ Возвращает расстояние от точки до начала координат "
        return ( self.x**2 + self.y**2 ) ** 0.5

p = Point2D()
p.x = 3
p.y = 4

p.r()  # 5
```

Неявную передачу self можно представлять как передачу параметра *p* в функцию *r()*, только этот параметр записывается не в скобках, а перед именем функции с точкой на конце.

Другие методы

Использование self похоже на явную передачу параметра в отдельную функцию.

```
class Point2D:
    x = 0
    y = 0
    # метод
    def r(self):
    """ Возвращает расстояние от точки до начала координат "
        return ( self.x**2 + self.y**2 ) ** 0.5
# функция
def R( point ):
    return ( point.x**2 + point.y**2 ) **2

p = Point2D()
p.x = 3
p.y = 4

p.r()  # 5

R( p )  # 5
```

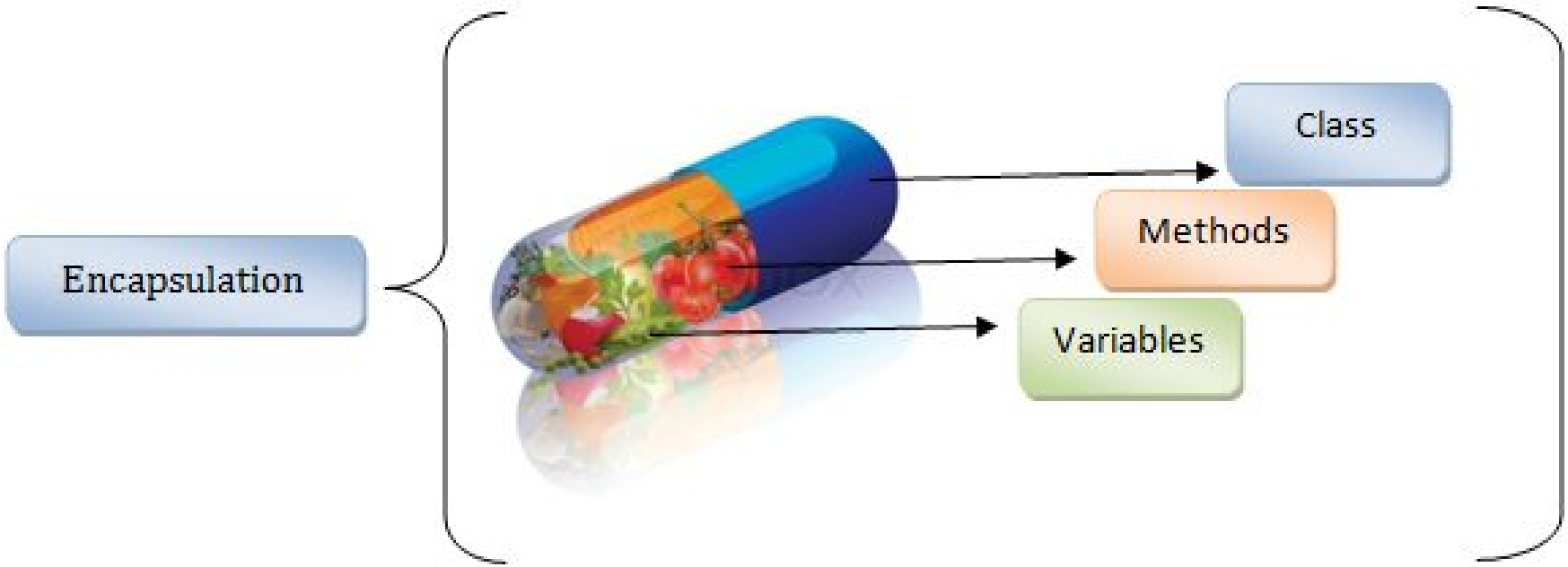
Инкапсуляция

Таким образом, помимо данных классы могут содержать ещё и методы работы с этими данными.

Такое объединение данных и методов к классе называется **инкапсуляцией**.

При этом подразумевается, что методы могут контролировать целостность и непротиворечивость данных.

Инкапсуляция



Пример

Класс "Точка"с двумя полями: x и y; двумя методами: конструктором __init__

```
class Point2D:
    x = 0
    y = 0

    #
    def __init__(self, x,y,z):
        self.x = x
        self.y = y

    def r(self):
    """ Возвращает расстояние от точки до начала координат """
        return ( self.x**2 + self.y**2 ) ** 0.5



p1 = Point2D( 10.5, -1.02 )
p2 = Point2D( 0, 42 )

p1.r()  # 10.55
p2.r()  # 42
```

- **Класс** - эти тип данных определяемый программистом.
- Класс может содержать как данные (**поля**) так и описывать операции над этими данными (**методы**).
- Объект - переменная имеющая тип определённого класса.

```
p = Point2D()

# p - объект
# Point2D - класс
```

- Объекты одного и того же класса имеют одинаковые методы и одинаковые поля, но данные содержащиеся в полях могут отличатся.
- Для обращения объекта к самому себе (своим полям и методам) используется зарезервированное имя **self**
- Все переменные в Python, включая простые типы, являются объектами.

Пользовательские (созданные программистом) объекты нельзя непосредственно выводить на экран, записывать введённые с клавиатуры данные непосредственно в объект.

Непосредственная запись в файл и чтение не рекомендуется, потому, что объект может содержать кроме данных ещё и методы.

Однако, для объекта можно написать методы, которые помогут выполнить вышеперечисленные операции.

Пример

```
class Point2D:
    x = 0
    y = 0
    # ...

    # преобразует массив байт в x,y
    def from_bytes(self, b):
        self.x, self.y = unpack('ff', b)

    # преобразует данные объекта в массив байт
    def to_bytes(self):
        return pack('ff', self.x, self.y)

p1 = Point2D(10.5, -1.02)
b = p1.to_bytes()
p1.from_bytes(b)
f = open('file.in','w+b')
f.write( b )
f.seek(0)
p1.from_bytes(f.read(8))
f.close()
```

Ссылки и литература

Ссылка на слайды

github.com/VetrovSV/Programming